Towards Verifiable Design Patterns

Dániel Petri

Design pattern [1] is a form for representing proven solutions in a predetermined way. It has three main parts, which describe the problem, the suggested solution and the consequences of applying the solution. Applying these *reusable* solutions to our problems allows us to avoid system development from the scratch and thus shortens the development cycle.

Design patterns are informal: they may contain class diagram fragments, sample source code lines and informal notes partitioned into several subsections. Their formalization can serve purposes like tool support, establishment of the relationships between two design patterns, investigation whether an implementation contains a pattern properly and so on. LePUS[5] [2] is a formal language for describing and reasoning about object oriented software architectures, designs and patterns. It is intended to describe relations between classes and functions. LePUS terms can be expressed both as diagrams and formulas.

The main goal is a tool, which identifies the appropriate design patterns that can be applied to a system under development. The design patterns are described in LePUS and are stored in a database. The system specification is given in UML Class Diagrams. The tool compares the specification with the existing patterns and selects the ones which (partially) fit in the specification. The output is a subset of design patterns with the additional information where each of the patterns fit in the specification.

The first step towards this tool is the transformation of the Class Diagrams into LePUS formalism. As most UML tools support XML export, the most evident way is to base the transformation on XML. This paper first describes the bijective transformation of LePUS diagrams into XML (called LePUS-XML) and the transformation of XML Class diagrams into LePUS-XML. This transformation uses XSLT that specifies rules by which one XML document is transformed into another.

In design patterns properties of the system can also be formulated, e.g. there is no single point of failure (SPoF) in it. If such patterns are implemented properly, statements can be made that the whole system is free of SPoF. The second task of my tool is to analyse whether such patterns are implemented properly.

References

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series, Addison-Wesley, Reading Mass. 1994.

[2] Amnon H. Eden: Formal Specification of Object Oriented Design, International Conference on Multidisciplinary Design in Engineering, CSME-MDE 2001, November 21-22, Montreal, Canada.

[5] Language for Patterns Uniform Specification